划重点 | 7种编译器的核心概念与算法

2020-08-07 朱英达

编译原理实战课 进入课程 >

你好，我是编辑王惠。

阶段性的总结复习和验证成果是非常重要的。所以，在 8 月 7 日到 8 月 12 日这为期一周的期中复习时间里，我们先来巩固一下“真实编译器解析篇”中的重点知识。你可以通过学习委员朱英达总结梳理的**划重点内容，以及涵盖了关键知识点的 7 张思维导图**，来回顾 7 种语言编译器的核心概念与算法。

另外，宫老师还精心策划了 10 道考试题，让你能在行至半程之时，做好自检，及时发现知识漏洞，到时候一起来挑战一下吧！

在期中复习周的最后，我还会邀请一位优秀的同学来做一次学习分享。通过他的学习故事，你也可以借此对照一下自己的编译原理学习之路。

好，下面我们就一起来复习这些核心的编译原理概念与算法知识吧。

Java 是一种广泛使用的计算机编程语言，主要应用于企业级 Web 应用开发、大型分布式系统以及移动应用开发（Android）。到现在，Java 已经是一门非常成熟的语言了，而且它也在不断进化、与时俱进，泛型、函数式编程、模块化等特性陆续都增加了进来。与此同时，Java 的编译器和虚拟机中所采用的技术，也比 20 年前发生了天翻地覆的变化。

Java 的字节码编译器（javac）是用 Java 编写的，它实现了自举。启动 Java 编译器需要 Java 虚拟机（默认是 HotSpot 虚拟机，使用 C++ 编写）作为宿主环境。

javac 编译器的编译过程，主要涉及到了这样一些关键概念和核心算法：

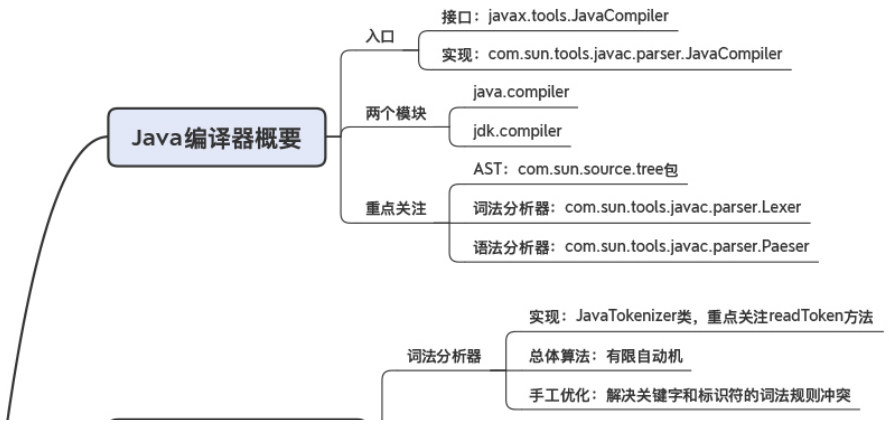
词法分析阶段：基于有限自动机的理论实现。在处理标识符与关键字重叠的问题上，采用了先都作为标识符识别出来，然后再把其中的关键词挑出来的方式。

语法分析阶段：使用了自顶向下的递归下降算法、LL(k) 方式以及多 Token 预读；处理左递归问题时，采用了标准的改写文法的方法；处理二元表达式时，采用了自底向上的运算符优先级解析器。

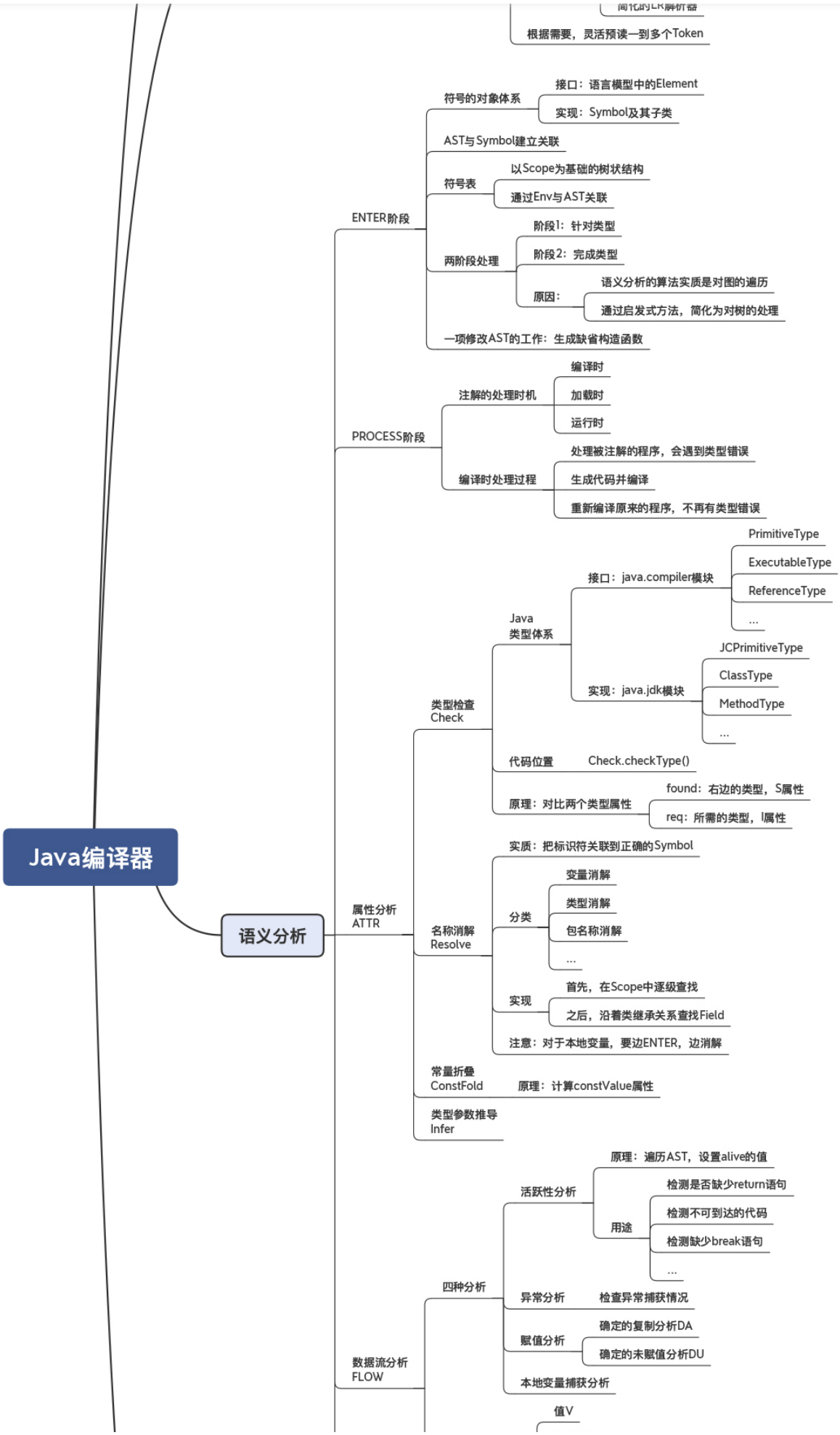
语义分析阶段：会分为多个小的阶段，且并不是顺序执行的，而是各阶段交织在一起。

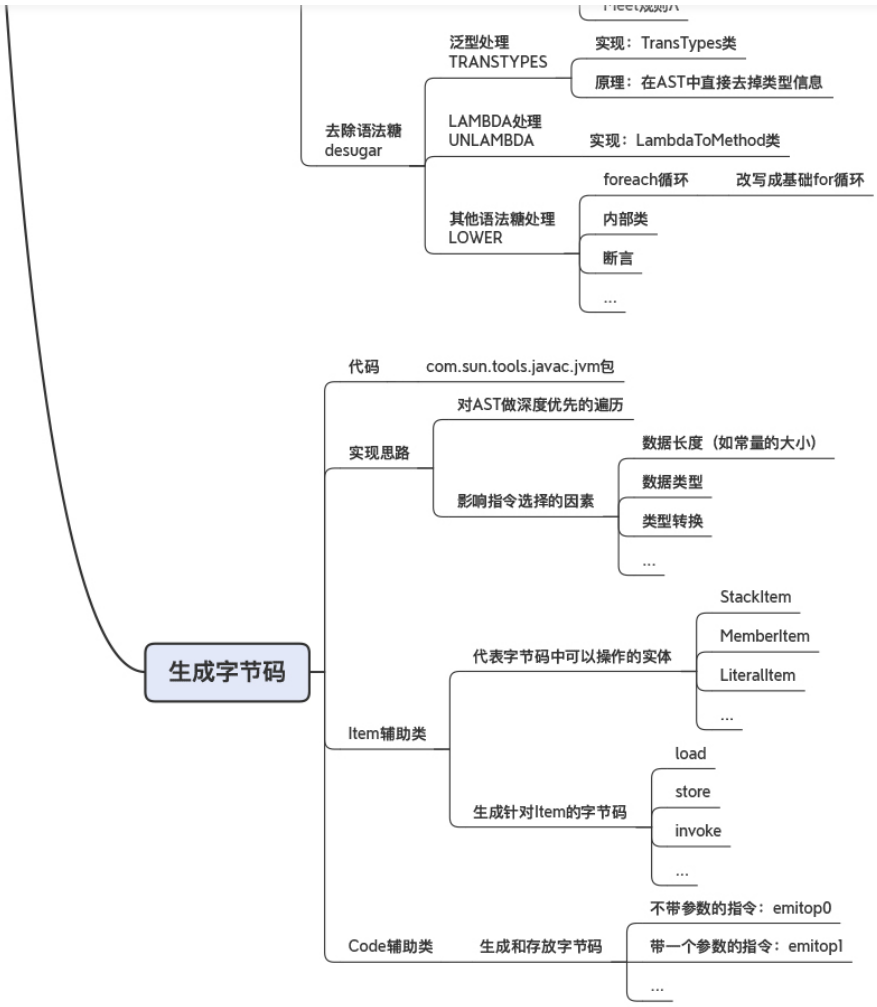
语义分析阶段主要包含：ENTER（建立符号表）、PROCESS（处理注解）、ATTR（属性分析）、FLOW（数据流分析）、TRANSTYPES（处理泛型）、TRANSPATTERNS（处理模式匹配）、UNLAMBDA（处理 Lambda）和 LOWER（处理其他所有的语法糖，比如内部类、foreach 循环等）、GENERATE 阶段（生成字节码）等。在 ATTR 和 FLOW 这两个阶段，编译器完成了主要的语义检查工作。

注意：生成字节码是一个比较机械的过程，编译器只需要对 AST 进行深度优先的遍历即可。在这个过程中会用到前几个阶段形成的属性信息，特别是类型信息。



下载APP


Java编译器
语义分析
属性分析
ATTR
ENTER阶段
符号的对象体系
接口：语言模型中的Element
实现：Symbol及其子类
AST与Symbol建立关联
符号表
以Scope为基础的树状结构
通过Env与AST关联
两阶段处理
阶段1：针对类型
阶段2：完成类型
原因：
语义分析的算法实质是对图的遍历
通过启发式方法，简化为对树的处理
一项修改AST的工作：生成缺省构造函数
PROCESS阶段
注解的处理时机
编译时
加载时
运行时
编译时处理过程
处理被注解的程序，会遇到类型错误
生成代码并编译
重新编译原来的程序，不再有类型错误
类型检查
Check
Java
类型体系
接口：java.compiler模块
PrimitiveType
ExecutableType
ReferenceType
...
实现：java.jdk模块
JCPrimitiveType
ClassType
MethodType
...
代码位置
Check.checkType()
原理：对比两个类型属性
found：右边的类型，S属性
req：所需的类型，I属性
名称消解
Resolve
实质：把标识符关联到正确的Symbol
分类
变量消解
类型消解
包名称消解
...
实现
首先，在Scope中逐级查找
之后，沿着类继承关系查找Field
注意：对于本地变量，要边ENTER，边消解
常量折叠
ConstFold
原理：计算constValue属性
类型参数推导
Infer
数据流分析
FLOW
四种分析
活跃性分析
原理：遍历AST，设置alive的值
用途
检测是否缺少return语句
检测不可到达的代码
检测缺少break语句
...
异常分析
检查异常捕获情况
赋值分析
确定的复制分析DA
确定的未赋值分析DU
本地变量捕获分析
值V
根据需要，灵活预读一到多个Token



极客时间

参考资料：

1. 关于注解的官方教程，参考这个链接。
2. 关于数据流分析的理论性内容，参考龙书（Compilers Principles, Techniques and Tools）第二版的 9.2 和 9.3 节。也可以参考《编译原理之美》 的第27、28讲，那里进行了比较直观的讲述。
3. 关于半格这个数学工具，可以参考龙书第二版的 9.3.1 部分，也可以参考《编译原理之美》的第 28 讲。
4. Java 语言规范第六章，参考Java 虚拟机指令集。

Java JIT 编译器（Graal）

做即时编译（JIT）。Java 以及基于 JVM 的语言，都受益于 JVM 的 JIT 编译器。

在JDK 的源代码中，你能找到 src/hotspot 目录，这是 JVM 的运行时：HotSpot 虚拟机，它是用 C++ 编写的，其中就包括 JIT 编译器。

Graal 是 Oracle 公司推出的一个完全用 Java 语言编写的 JIT 编译器。Graal 编译器有两个特点：内存安全（相比 C++ 实现的 Java JIT 编译器而言）；与 Java 配套的各种工具（比如 ID）更友好、更丰富。

Java JIT 编译器的编译过程，主要涉及到了这样一些关键概念和核心算法：

分层编译：C0（解释器）、C1（客户端编译器）、C2（服务端编译器）。不同阶段的代码优化激进的程度不同，且存在 C2 降级回 C1 的逆优化。

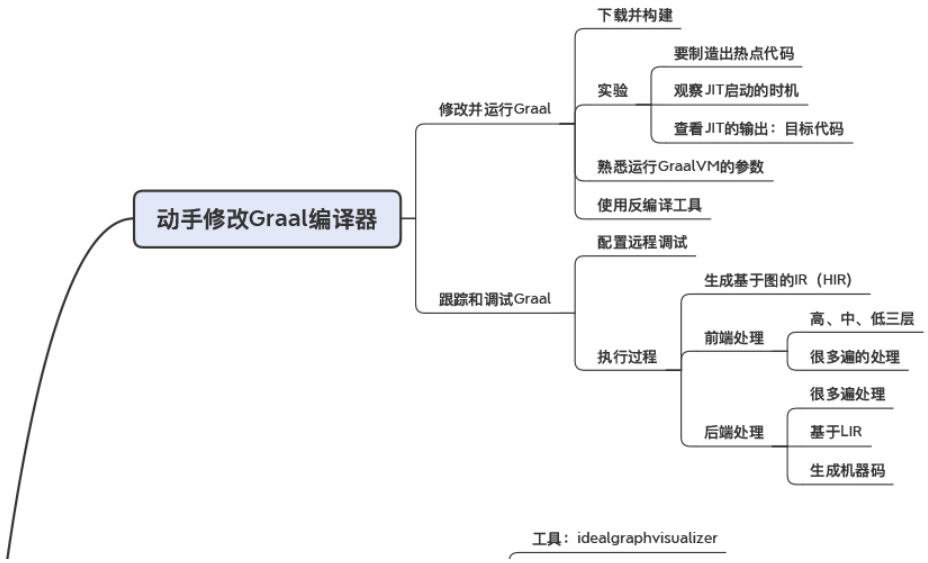
IR 采用了“节点之海（Sea of Nodes）”，整合了控制流图与数据流图，符合 SSA 格式，有利于优化算法的编写和维护。

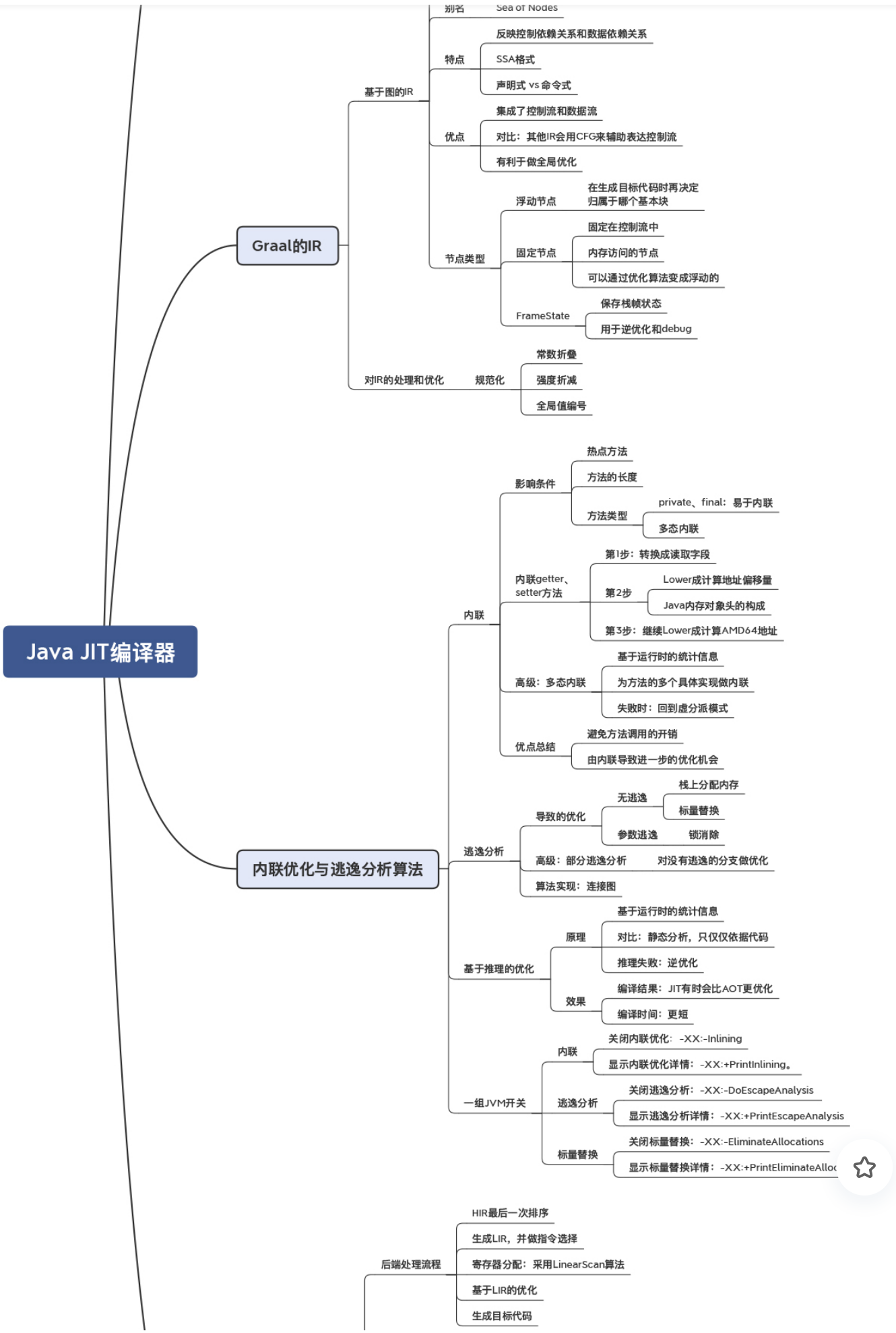
两个重要的优化算法：内联优化和逃逸分析。

几个重要的数据结构：HIR（硬件无关的 IR）、LIR（硬件相关的 IR）、CFG（控制流图）。

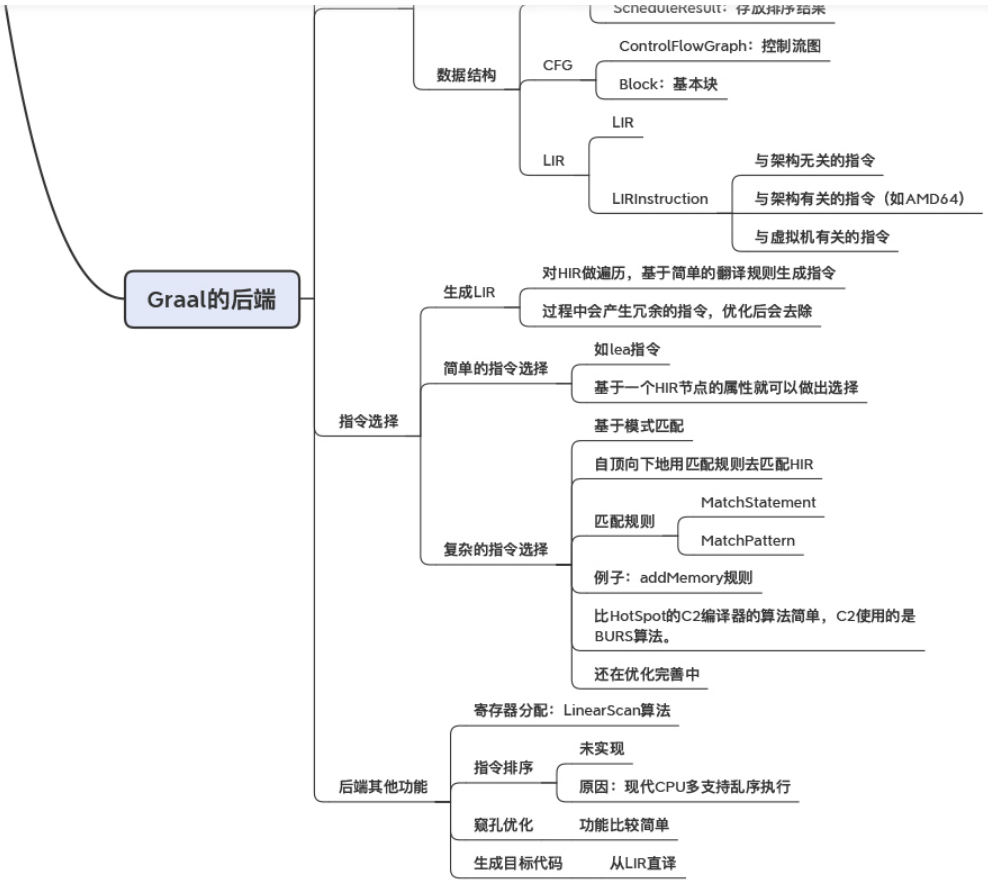
寄存器分配算法：LinearScan。

金句摘录：“编译器开发的真正的工作量，都在中后端。”




Java JIT编译器
Graal的IR
基于图的IR
别名
Sea of Nodes
特点
反映控制依赖关系和数据依赖关系
SSA格式
声明式 vs 命令式
优点
集成了控制流和数据流
对比：其他IR会用CFG来辅助表达控制流
有利于做全局优化
节点类型
浮动节点
在生成目标代码时再决定归属于哪个基本块
固定节点
固定在控制流中
内存访问的节点
可以通过优化算法变成浮动的
FrameState
保存栈帧状态
用于逆优化和debug
对IR的处理和优化
规范化
常数折叠
强度折减
全局值编号
内联优化与逃逸分析算法
内联
影响条件
热点方法
方法的长度
方法类型
private、final：易于内联
多态内联
内联getter、setter方法
第1步：转换成读取字段
第2步
Lower成计算地址偏移量
Java内存对象头的构成
第3步：继续Lower成计算AMD64地址
高级：多态内联
基于运行时的统计信息
为方法的多个具体实现做内联
失败时：回到虚分派模式
优点总结
避免方法调用的开销
由内联导致进一步的优化机会
逃逸分析
导致的优化
无逃逸
栈上分配内存
标量替换
参数逃逸
锁消除
高级：部分逃逸分析
对没有逃逸的分支做优化
算法实现：连接图
基于推理的优化
原理
基于运行时的统计信息
对比：静态分析，只仅仅依据代码
推理失败：逆优化
效果
编译结果：JIT有时会比AOT更优化
编译时间：更短
一组JVM开关
内联
关闭内联优化：-XX:-Inlining
显示内联优化详情：-XX:+PrintInlining。
逃逸分析
关闭逃逸分析：-XX:-DoEscapeAnalysis
显示逃逸分析详情：-XX:+PrintEscapeAnalysis
标量替换
关闭标量替换：-XX:-EliminateAllocations
显示标量替换详情：-XX:+PrintEliminateAlloc
后端处理流程
HIR最后一次排序
生成LIR，并做指令选择
寄存器分配：采用LinearScan算法
基于LIR的优化
生成目标代码

下载APP



极客时间

参考资料：

1. GraalVM 项目的官方网站；Graal 的Github 地址；Graal 项目的出版物。
2. 基于图的 IR 的必读论文：程序依赖图 - J. Ferrante, K. J. Ottenstein, and J. D. Warren. The program dependence graph and its use in optimization. July 1987；Click 的论文 - A Simple Graph-Based Intermediate Representation；介绍 Graal IR 的论文 - Graal IR: An Extensible Declarative Intermediate Representation。
3. 关于优化算法：多态内联 - Inlining of Virtual Methods；逃逸分析 - Escape Analysis for Java；部分逃逸分析 - Partial Escape Analysis and Scalar Replacement for Java。

Python 编译器（CPython）

Python 诞生于上个世纪 90 年代初，作者是荷兰计算机程序员吉多·范罗苏姆（Guido van Rossum）。Python 语言的特点是：自身语法简单，容易掌握，强调一件事情只能用一种

编写自己的 C++ 扩展，实现一些高性能模块。

另外，Python 使用了 pgen 这样的生成编译器的工具。pgen 能够基于语法规则生成解析表（Parse Table），供语法分析程序使用。你可以通过修改规则文件来修改 Python 语言的语法，pgen 能给你生成新的语法解析器。它是把 EBNF 转化成一个 NFA，然后再把这个 NFA 转换成 DFA。基于这个 DFA，在读取 Token 的时候，编译器就知道如何做状态迁移，并生成解析树。Python 用的是 LL(1) 算法。

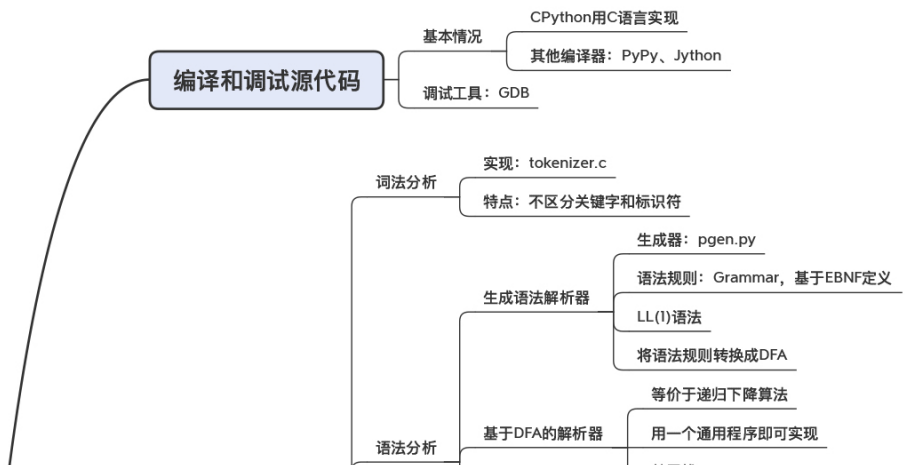
CPython 编译器编译器的编译过程，主要涉及到了这样一些关键概念和核心算法：

语法分析：首先是生成 CST（Concret Syntax Tree，具体语法树），接着生成 AST（抽象语法树）。CST 的特点是完全基于源程序结构构建出树结构，它比 AST 啰嗦，但更精确地反映了语法推导的过程。而 AST 的特点是去掉了树结构上繁杂冗余的树枝节点，更简明扼要，更准确地表达了程序的结构。

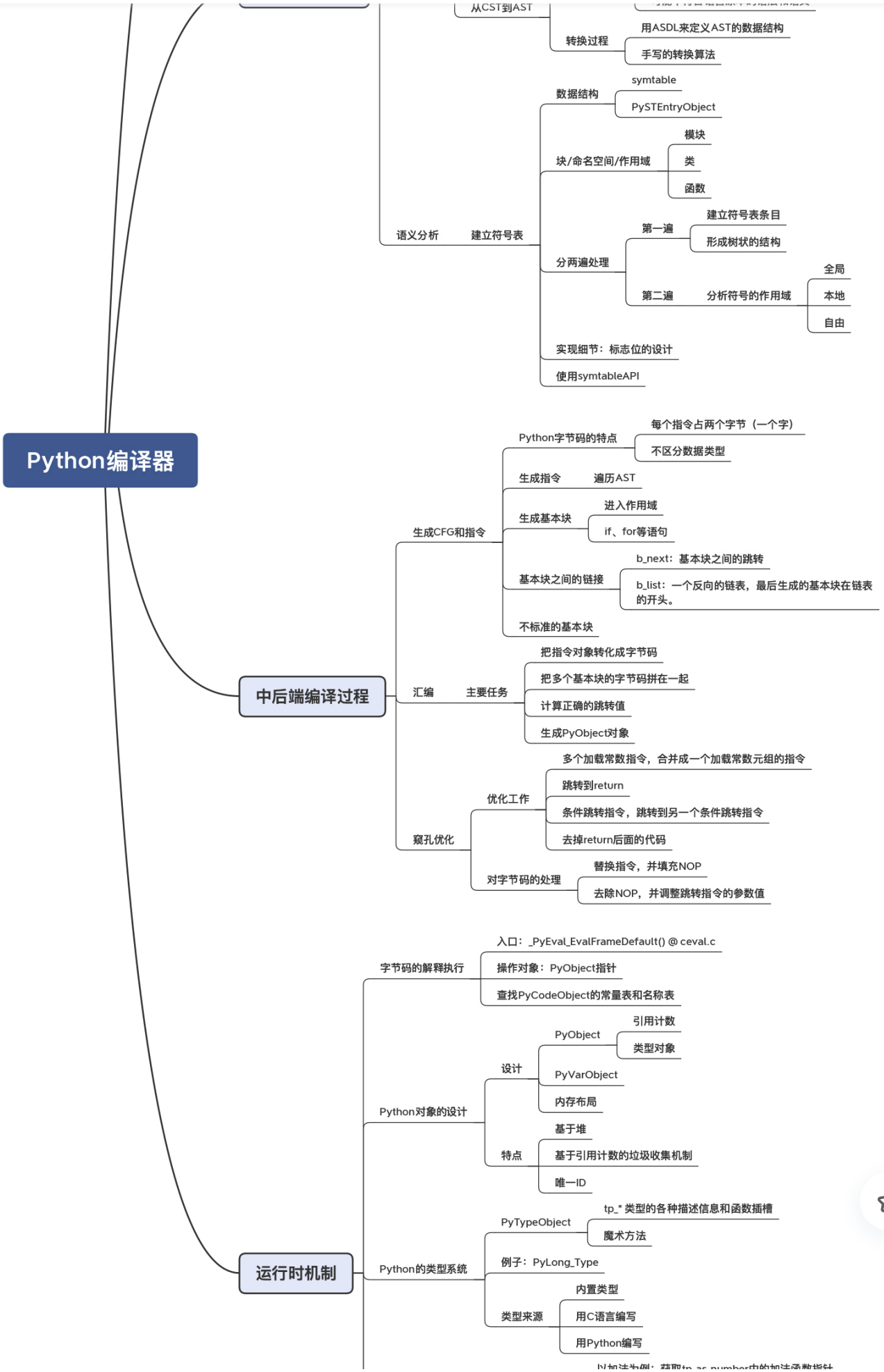
语义分析：Python 通过一个建立符号表的过程来做相关的语义分析，包括做引用消解等。Python 语言使用变量的特点是不存在变量声明，每个变量都是赋值即声明，因此在给一个变量赋值时需要检查作用域，确认当前操作是给全局的变量赋值，还是在局部给声明新变量。

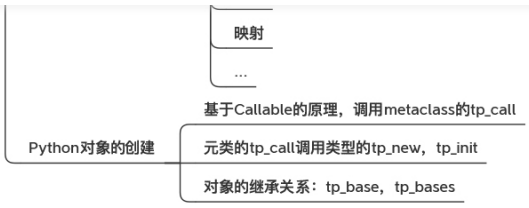
生成字节码：这个工作实际上包含了生成 CFG、为每个基本块生成指令，以及把指令汇编成字节码，并生成 PyCodeObject 对象的过程。另外，生成的字节码在最后放到解释器里执行之前，编译器还会再做一步窥孔优化工作。

运行时机制：Python 的运行时设计的核心，就是 PyObject 对象，Python 对象所有的特性都是从 PyObject 的设计中延伸出来的。其虚拟机采用了栈机的架构。



下载APP


Python编译器
中后端编译过程
从CST到AST
转换过程
用ASDL来定义AST的数据结构
手写的转换算法
数据结构
symtable
PySTEntryObject
块/命名空间/作用域
模块
类
函数
语义分析
建立符号表
分两遍处理
第一遍
建立符号表条目
形成树状的结构
第二遍
分析符号的作用域
全局
本地
自由
实现细节：标志位的设计
使用symtableAPI
生成CFG和指令
Python字节码的特点
每个指令占两个字节（一个字）
不区分数据类型
生成指令
遍历AST
生成基本块
进入作用域
if、for等语句
基本块之间的链接
b_next：基本块之间的跳转
b_list：一个反向的链表，最后生成的基本块在链表的开头。
不标准的基本块
汇编
主要任务
把指令对象转化成字节码
把多个基本块的字节码拼在一起
计算正确的跳转值
生成PyObject对象
窥孔优化
优化工作
多个加载常数指令，合并成一个加载常数元组的指令
跳转到return
条件跳转指令，跳转到另一个条件跳转指令
去掉return后面的代码
对字节码的处理
替换指令，并填充NOP
去除NOP，并调整跳转指令的参数值
运行时机制
字节码的解释执行
入口：_PyEval_EvalFrameDefault() @ ceval.c
操作对象：PyObject指针
查找PyCodeObject的常量表和名称表
Python对象的设计
设计
PyObject
引用计数
类型对象
PyVarObject
内存布局
特点
基于堆
基于引用计数的垃圾收集机制
唯一ID
Python的类型系统
PyTypeObject
tp_* 类型的各种描述信息和函数插槽
魔术方法
例子：PyLong_Type
类型来源
内置类型
用C语言编写
用Python编写



极客时间

参考资料：

1. python.org 网站：下载3.8.1 版本的源代码。
2. GDB 的安装和配置：参考这篇文章。
3. Python 的开发者指南网站。
4. pgen 的工具程序：Parser/pgen。
   注：由于 CPython 最新的 Master 分支上的代码调整，此处 pgen 的链接地址调整为 CPython3.9 版本分支上的 pgen 相关代码。
5. Python 的字节码的说明。
6. Python 的内置类型。

JavaScript 编译器（V8）

V8 是谷歌公司在 2008 年推出的一款 JavaScript 编译器，主要由 C++ 编写而成。V8 主要应用于 Chrome 浏览器，后来也被开源社区中诸如 Node.js 等项目所使用。其最为突出的特点就是“快”，由于 JavaScript 是在浏览器下载完页面后马上编译并执行，它对编译速度有更高的要求。因此，V8 采用了一系列技术手段优化编译和启动阶段运行速度。

在设计上，V8 结合了分阶段懒解析、空间换时间等设计思路，突出了解析、启动阶段运行的时间开销。

对源代码的 Parse，进行了流（Stream）化处理，也就是边下载边解析。

预解析（PreParse）处理，也就是所谓懒解析最核心的设计思想，每个 JS 函数只有被执行的时候才会解析函数体的内部逻辑。

分组成，类似于 Java 编译器的字节码解释执行和 Graal 优化编译后执行两阶段；TurboFan 编译器的 IR 也采用了 Sea of Nodes，这一点类似于 Java 的 Graal 编译器，且也涉及到了内联优化和逃逸分析算法。

其运行方式分为两类：

常规情况下，Ignition 字节码解释执行；

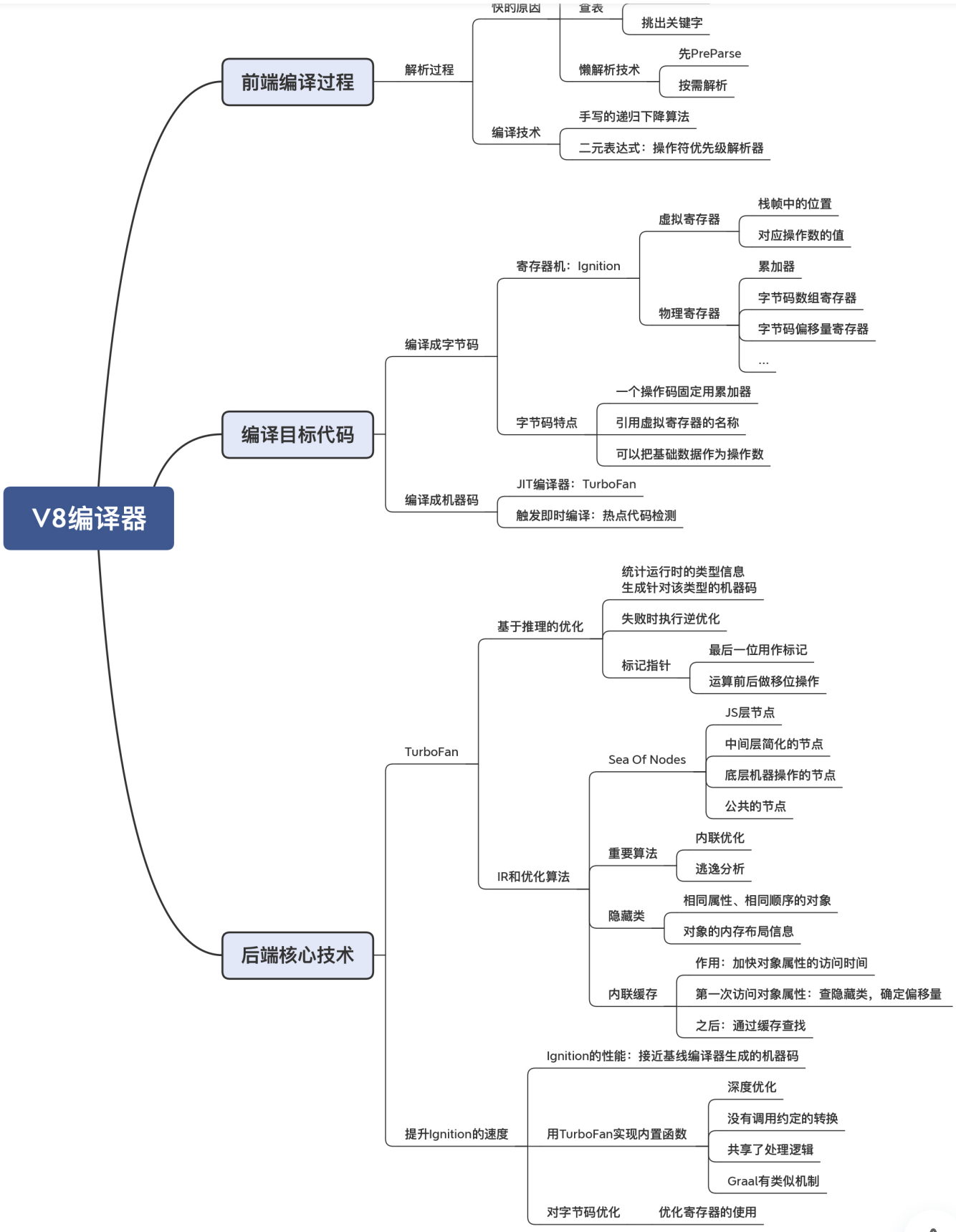
编译器判定热点代码，TurboFan JIT 编译成机器码执行，并且 TurboFan 会依赖一些 Ignition 解释执行过程中的运行时数据，来进行进一步优化，使机器码尽可能高效。

因为 JavaScript 是动态类型语言，因此对函数参数类型的推断以及针对性优化是一个 V8 的核心技术。V8 涉及到的其他优化算法有：

隐藏类（Hidden Class）。相同“形状”的 JS 对象会被以同一个隐藏类维护其数据结构。

内联缓存（Inline Caching）。针对隐藏类查找属性值时的指针偏移量，进行内联缓存，这属于结合隐藏类技术做进一步性能的优化。

下载APP


V8编译器
前端编译过程
解析过程
快的原因
查表
挑出关键字
懒解析技术
先PreParse
按需解析
编译技术
手写的递归下降算法
二元表达式：操作符优先级解析器
编译目标代码
编译成字节码
寄存器机：Ignition
虚拟寄存器
栈帧中的位置
对应操作数的值
物理寄存器
累加器
字节码数组寄存器
字节码偏移量寄存器
...
字节码特点
一个操作码固定用累加器
引用虚拟寄存器的名称
可以把基础数据作为操作数
编译成机器码
JIT编译器：TurboFan
触发即时编译：热点代码检测
后端核心技术
TurboFan
基于推理的优化
统计运行时的类型信息
生成针对该类型的机器码
失败时执行逆优化
标记指针
最后一位用作标记
运算前后做移位操作
IR和优化算法
Sea Of Nodes
JS层节点
中间层简化的节点
底层机器操作的节点
公共的节点
重要算法
内联优化
逃逸分析
隐藏类
相同属性、相同顺序的对象
对象的内存布局信息
内联缓存
作用：加快对象属性的访问时间
第一次访问对象属性：查隐藏类，确定偏移量
之后：通过缓存查找
提升Ignition的速度
Ignition的性能：接近基线编译器生成的机器码
用TurboFan实现内置函数
深度优化
没有调用约定的转换
共享了处理逻辑
Graal有类似机制
对字节码优化
优化寄存器的使用

参考资料：

the scanner；Blazingly fast parsing, part 2: lazy parsing。

3. 了解 Ignition 的设计：Ignition Design Doc，宫老师在 Github 上也放了一个拷贝。
4. 了解 Ignition 的字节码：Understanding V8’ s bytecode。
5. V8 的指针压缩技术：Pointer Compression in V8。
6. 介绍 V8 基于推理的优化机制：An Introduction to Speculative Optimization in V8。
7. 关于 Ignition 字节码做优化的论文：Register equivalence optimization，宫老师在 Github 上也放了一份拷贝。

Julia 的编译器

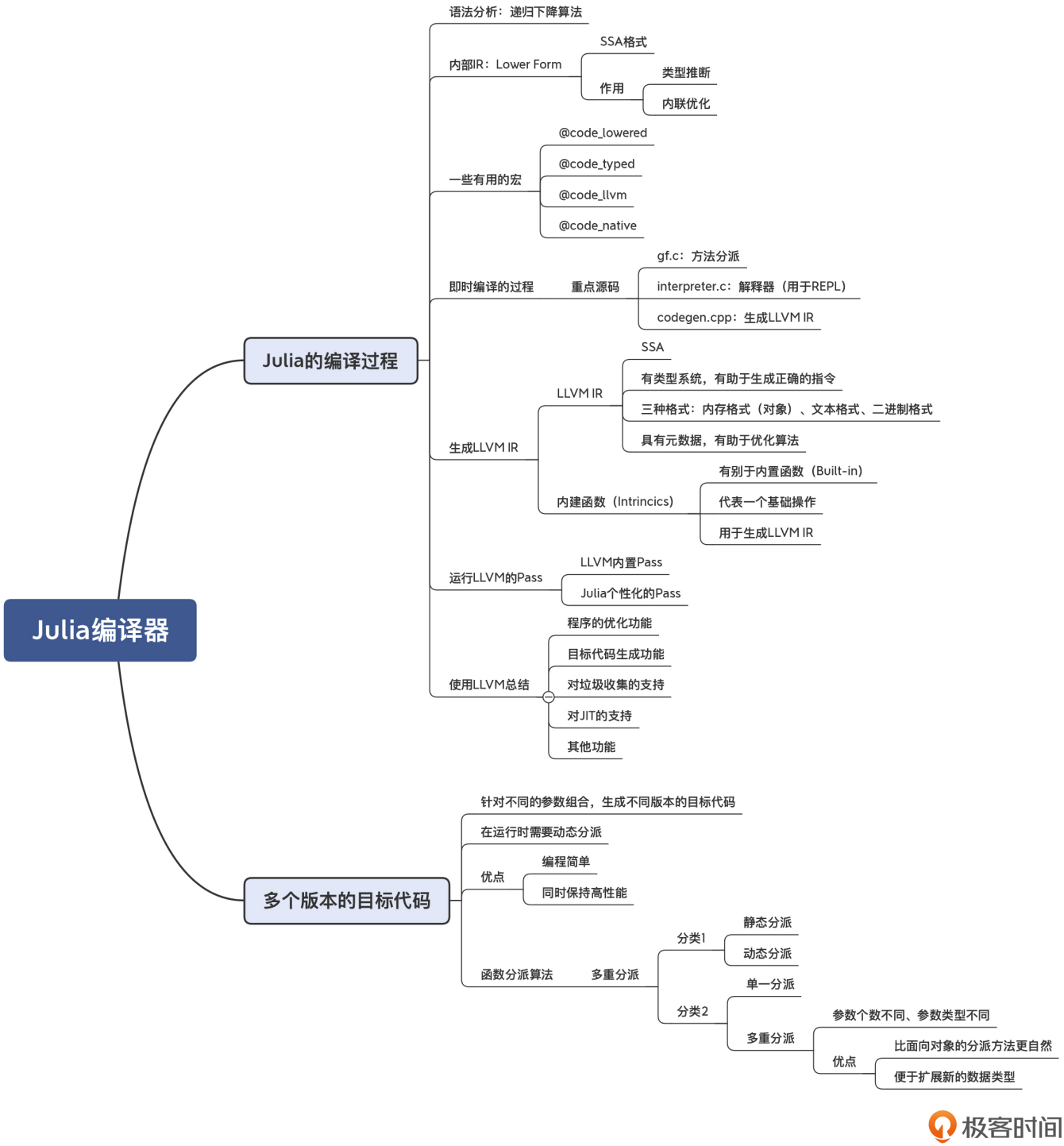
Julia 语言最初发行于 2012 年，其最初是为了满足高性能数值分析和计算科学的需要而设计的。Julia 同时兼具了静态编译型和动态解释型语言的优点：一方面它的性能很高，可以跟 Java 和 C 语言媲美；另一方面，它又是动态类型的，编写程序时不需要指定类型。

Julia 编译器的特点是：

> 作为动态类型语言，却能拥有静态类型语言的执行速度，最关键的原因是使用了 LLVM 作为编译器后端，针对动态类型的变量，在运行时由编译器 JIT 编译生成多个版本的目标代码，保证程序的高性能；
>
> 由 C、C++、Lisp 和 Julia 四种语言开发而成，编译器前端主要采用 Lisp 实现；
>
> 考虑到对函数式编程的支持，有别于面向对象编程的“单一分派”，Julia 的编译器提供了“多重分派”的功能。

Julia 编译器的编译过程，主要涉及到了这样一些关键概念和核心算法：

> Julia 的编译器也是采用了递归下降算法来实现语法分析。
>
> 其内部 IR 采用了 SSA 格式，主要作用是完成类型推断和内联优化。



参考资料：

1. LLVM 的官网，以及 LLVM 的源代码。
2. Julia 的开发者文档中有对如何使用 LLVM 的介绍：Working with LLVM。
3. 对 LLVM 中的各种 Pass 的介绍：LLVM’ s Analysis and Transform Passes。

该如何生成 LLVM IR，最后即时编译并运行。你可以参考一下。

Go 语言编译器（gc）

Go 语言是 Google 开发的一种静态强类型、编译型、并发型，并具有垃圾回收功能的编程语言，又名 Golang。Go 广泛应用于 Google 的产品以及许多其他组织和开源项目，其创建的初衷就是主要面向于部署于大量服务器之间的分布式程序，也就是我们今天所说的“云”。因此，Go 的主要优势聚焦于服务端高并发场景。

Go 语言编译器的特点是：

gc 编译器除了少量标准库的内容是用汇编写的以外，其绝大部分逻辑，都是用 Go 语言本身写的，因此实现了较为完整的自举（Bootstraping），从前端到后端的整个流程都使用 Go 语言实现，在编程语言中是较为罕见的；

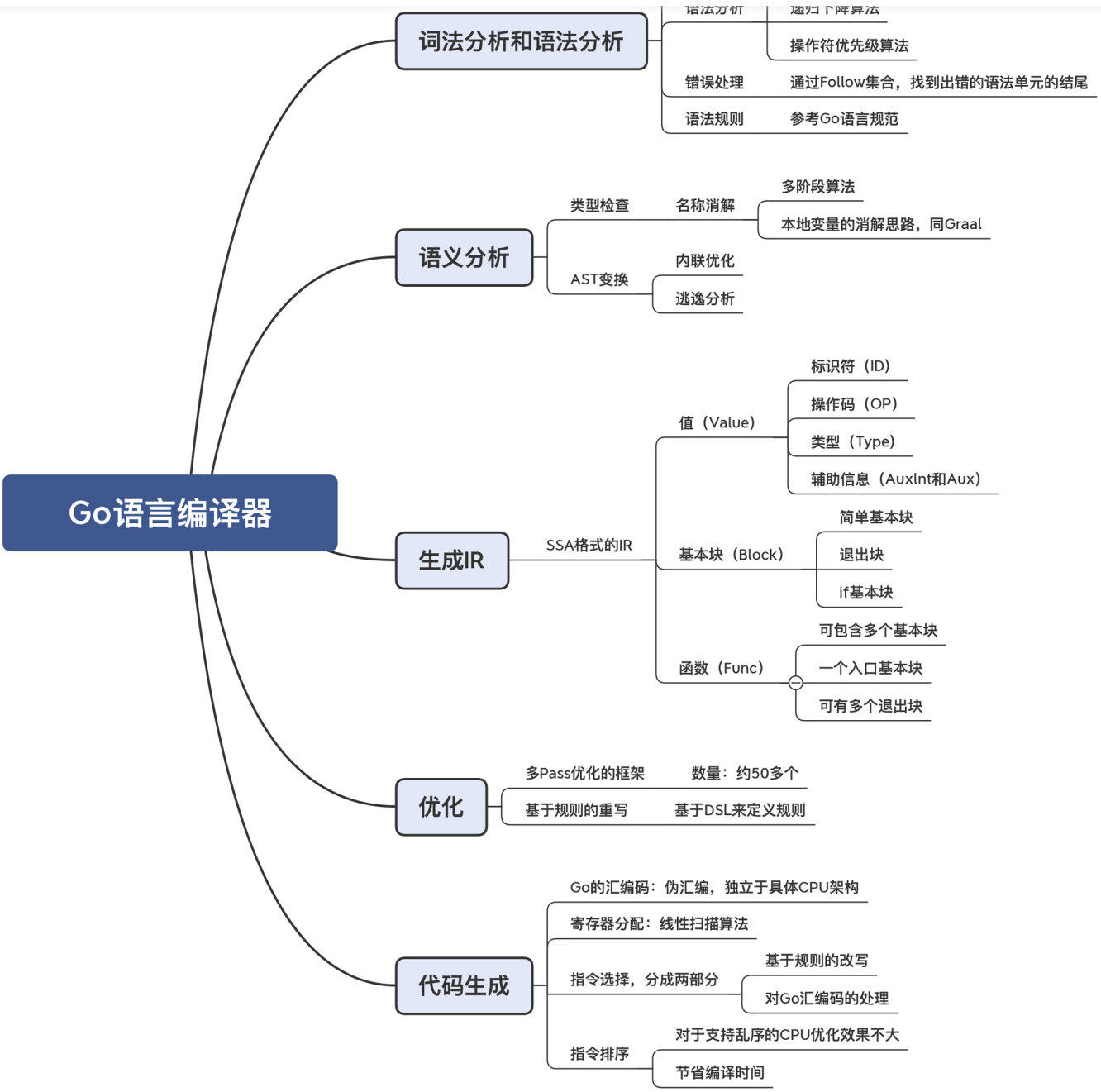
教科书级别的设计，源代码完全自举、代码干净利索，因此非常适合作为学习参考。

Go 语言编译器的编译过程，主要涉及到了这样一些关键概念和核心算法：

编译器前端：gc 编译器的词法分析和语法分析使用的都是非常经典、传统的算法，如手写的递归下降算法、处理二元表达式时使用操作符优先级算法。

中间代码阶段：SSA 格式的 IR；基于 CFG 的 IR，利于“死代码”的发现与删除；多遍（Pass）的优化框架。

机器码生成阶段：线性扫描算法；官方的 gc 编译器并没做指令重排，这是基于编译过程中时间开销的考虑。



参考资料：

1. 介绍 gc 编译器的主要结构：Introduction to the Go compiler官方文档。
2. 介绍 gc 编译器的 SSA：Introduction to the Go compiler’ s SSA backend官方文档。
3. Go compiler internals: adding a new statement to Go - Part 1、Part2。在这两篇博客里，作者做了一个实验：如果往 Go 里面增加一条新的语法规则，需要做哪些事情。我们能够很好地、贯穿性地了解一个编译器的方法。

5. 介绍 Go 汇编的细节：A Primer on Go Assembly和A Quick Guide to Go’s Assembler。gc 编译器采用的汇编语言是它自己的一种格式，是“伪汇编”。

MySQL 的编译器

MySQL 是一个开放源码的关系数据库管理系统，原开发者为瑞典的 MySQL AB 公司，后几经辗转，目前归属于 Oracle 旗下产品。在过去，MySQL 性能高、成本低、可靠性好，因此成为了最流行的开源数据库。SQL 可以称得上是最成功的 DSL（特定领域语言）之一，MySQL 中的 SQL 解析模块则是这门 DSL 的非常具有可参考性的一个实现。MySQL 使用 C++ 编写，有少量几个代码文件是用 C 语言编写的。

MySQL 的编译器的特点是：

SQL 作为 DSL 中最具有代表性的一种语言，学习它的编译器的实现，可以为我们自己设计面向业务的 DSL 提供参考。

手写的词法分析、用工具（bison，GNU 的 yacc）生成的语法分析。

基于 LALR 处理二元表达式。

中后端优化方面：MySQL 是解释执行，并没有做太多的机器相关优化；在机器无关优化方面，除了一些编译领域传统的优化技术之外，还做了一些针对数据库特定场景的优化方式。

MySQL 的编译器的编译过程，主要涉及到了这样一些关键概念和核心算法：

词法分析和语法分析

词法分析：手写的词法分析器。

语法分析：由 bison 生成。bison 是一种基于 EBNF 生成语法分析程序的工具，可视为 GNU 版的 yacc。

bison 支持的语法分析算法：LALR 算法。

语义分析

基于数据库“业务逻辑”的引用消解：库名、表名、列名、入口、查找和作用域（子查询）

机器无关优化

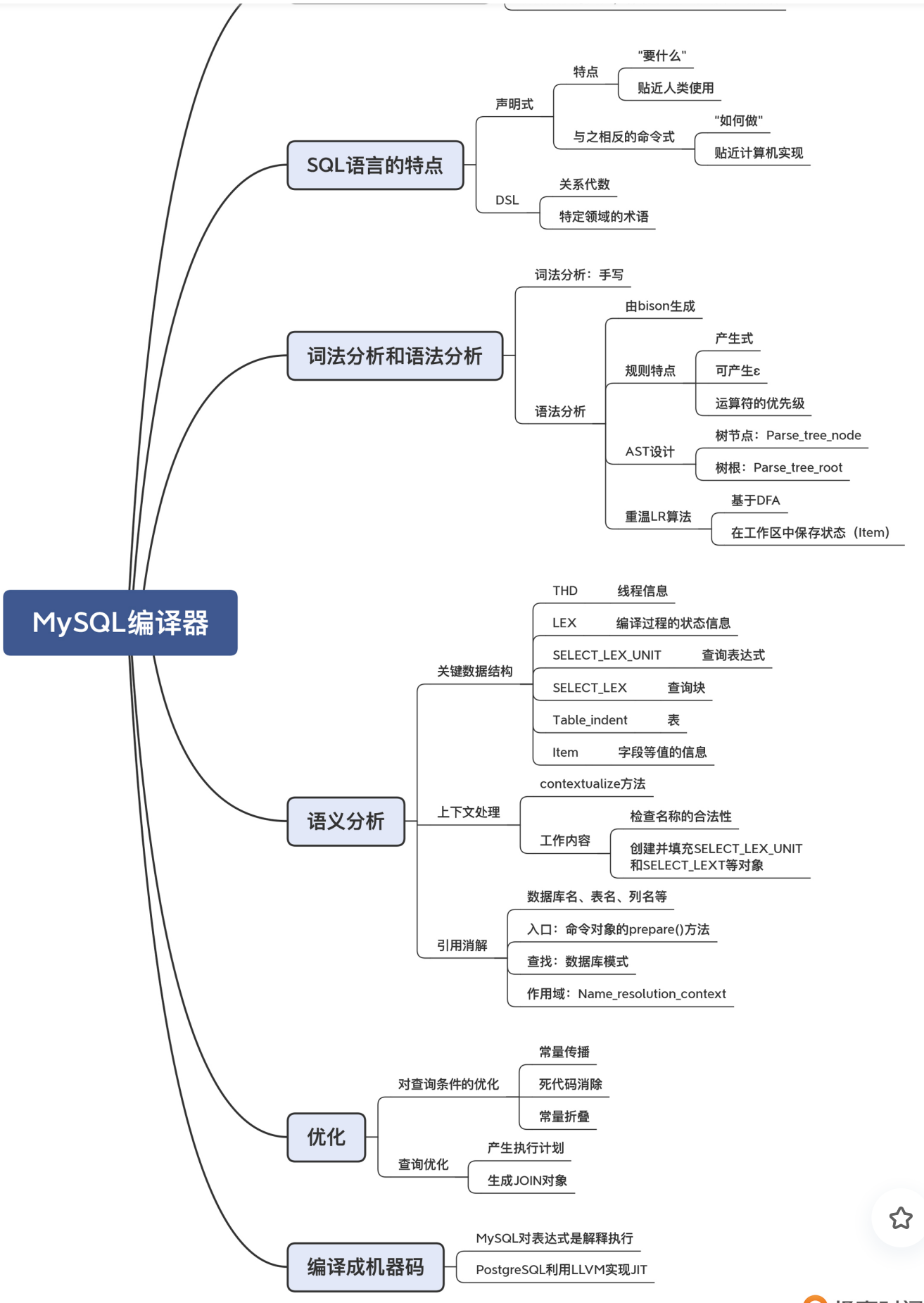
典型的优化：常量传播、死代码消除、常量折叠。

针对数据库场景的优化：产生执行计划、生成 JOIN 对象。

机器相关优化

MySQL 实际上对表达式是解释执行，所以并没有真正意义上做任何机器相关优化。

列举了 PG 另一种类似 MySQL 的 DB，有通过 LLVM 的 JIT 优化。


MySQL编译器
SQL语言的特点
声明式
特点
"要什么"
贴近人类使用
与之相反的命令式
"如何做"
贴近计算机实现
DSL
关系代数
特定领域的术语
词法分析和语法分析
词法分析：手写
语法分析
由bison生成
规则特点
产生式
可产生ε
运算符的优先级
AST设计
树节点：Parse_tree_node
树根：Parse_tree_root
重温LR算法
基于DFA
在工作区中保存状态（Item）
语义分析
关键数据结构
THD 线程信息
LEX 编译过程的状态信息
SELECT_LEX_UNIT 查询表达式
SELECT_LEX 查询块
Table_indent 表
Item 字段等值的信息
上下文处理
contextualize方法
工作内容
检查名称的合法性
创建并填充SELECT_LEX_UNIT和SELECT_LEXT等对象
引用消解
数据库名、表名、列名等
入口：命令对象的prepare()方法
查找：数据库模式
作用域：Name_resolution_context
优化
对查询条件的优化
常量传播
死代码消除
常量折叠
查询优化
产生执行计划
生成JOIN对象
编译成机器码
MySQL对表达式是解释执行
PostgreSQL利用LLVM实现JIT

极客时间

1. 下载MySQL 的源代码；跟踪 MySQL 的执行过程，要用 Debug 模式编译 MySQL，具体步骤可以参考这篇开发者文档。
2. MySQL 的内行手册：MySQL Internals Manual。它能给我们提供一些重要的信息，但文档内容经常跟源代码的版本不同步，比如介绍源代码的目录结构的信息就过时了。需要注意一下。
3. bison 的手册。
4. 如果要加深对 MySQL 内部机制的了解，宫老师推荐了两本书：OReilly 的《Understanding MySQL Internals》，以及《Expert MySQL》。

提建议

更多课程推荐

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

下一篇 期中考试 | 这些编译原理知识，你都掌握了吗？

精选留言 (1)

写留言

Dr.z
2020-08-10

宫老师 你好， 在Python编译器那部分的超链接：pgen 的工具程序：Parser/pgen。好像失效了，有可以替代的链接吗？

展开

编辑回复: 链接已更换~~因为CPython最新的Master分支上的代码调整，此处pgen的链接地址调整为了CPython3.9版本分支上的pgen相关代码~

1